

## Section 10 - Domain, host, address, and local part lists

### 10. Domain, host, address, and local part lists

A number of Exim configuration options contain lists of domains, hosts, email addresses, or local parts. For example, the `hold_domains` option contains a list of domains whose delivery is currently suspended. These lists are also used as data in ACL statements (see chapter 39), and as arguments to expansion conditions such as `match_domain`.

Each item in one of these lists is a pattern to be matched against a domain, host, email address, or local part, respectively. In the sections below, the different types of pattern for each case are described, but first we cover some general facilities that apply to all four kinds of list. 10.1 Expansion of lists

Each list is expanded as a single string before it is used. The result of expansion must be a list, possibly containing empty items, which is split up into separate items for matching. By default, colon is the separator character, but this can be varied if necessary. See sections 6.19 and 6.20 for details of the list syntax; the second of these discusses the way to specify empty list items.

If the string expansion is forced to fail, Exim behaves as if the item it is testing (domain, host, address, or local part) is not in the list. Other expansion failures cause temporary errors.

If an item in a list is a regular expression, backslashes, dollars and possibly other special characters in the expression must be protected against misinterpretation by the string expander. The easiest way to do this is to use the `\N` expansion feature to indicate that the contents of the regular expression should not be expanded. For example, in an ACL you might have:

```
deny senders = \N^d{8}\w@.*\baddomain\example$\N : \
    ${lookup{$domain}!search{/badsenders/bydomain}}
```

The first item is a regular expression that is protected from expansion by `\N`, whereas the second uses the expansion to obtain a list of unwanted senders based on the receiving domain. 10.2 Negated items in lists

Items in a list may be positive or negative. Negative items are indicated by a leading exclamation mark, which may be followed by optional white space. A list defines a set of items (domains, etc). When Exim processes one of these lists, it is trying to find out whether a domain, host, address, or local part (respectively) is in the set that is defined by the list. It works like this:

The list is scanned from left to right. If a positive item is matched, the subject that is being checked is in the set; if a negative item is matched, the subject is not in the set. If the end of the list is reached without the subject having matched any of the patterns, it is in the set if the last item was a negative one, but not if it was a positive one. For example, the list

```
in domainlist relay_domains = !a.b.c : *.b.c
```

matches any domain ending in `.b.c` except for `a.b.c`. Domains that match neither `a.b.c` nor `*.b.c` do not match, because the last item in the list is positive. However, if the setting were

```
domainlist relay_domains = !a.b.c
```

then all domains other than `a.b.c` would match because the last item in the list is negative. In other words, a list that ends with a negative item behaves as if it had an extra item `!*` on the end.

Another way of thinking about positive and negative items in lists is to read the connector as `&ldquo;or&rdquo;` after a positive item and as `&ldquo;and&rdquo;` after a negative item. 10.3 File names in lists

If an item in a domain, host, address, or local part list is an absolute file name (beginning with a slash character), each line of the file is read and processed as if it were an independent item in the list, except that further file names are not allowed, and no expansion of the data from the file takes place. Empty lines in the file are ignored, and the file may also contain comment lines:

-

For domain and host lists, if a `#` character appears anywhere in a line of the file, it and all following characters are ignored.

-

Because local parts may legitimately contain `#` characters, a comment in an address list or local part list file is recognized only if `#` is preceded by white space or the start of the line. For example: `not#comment@x.y.z #` but this is a comment

Putting a file name in a list has the same effect as inserting each line of the file as an item in the list (blank lines and comments excepted). However, there is one important difference: the file is read each time the list is processed, so if its contents vary over time, Exim's behaviour changes.

If a file name is preceded by an exclamation mark, the sense of any match within the file is inverted. For example, if `hold_domains = !/etc/nohold-domains`

and the file contains the lines `!a.b.c`  
`*.b.c`

then `a.b.c` is in the set of domains defined by `hold_domains`, whereas any domain matching `*.b.c` is not. 10.4 An lsearch file is not an out-of-line list

As will be described in the sections that follow, lookups can be used in lists to provide indexed methods of checking list membership. There has been some confusion about the way lsearch lookups work in lists. Because an lsearch file contains plain text and is scanned sequentially, it is sometimes thought that it is allowed to contain wild cards and other kinds of non-constant pattern. This is not the case. The keys in an lsearch file are always fixed strings, just as for any other single-key lookup type.

If you want to use a file to contain wild-card patterns that form part of a list, just give the file name on its own, without a search type, as described in the previous section. 10.5 Named lists

A list of domains, hosts, email addresses, or local parts can be given a name which is then used to refer to the list elsewhere in the configuration. This is particularly convenient if the same list is required in several different places. It also allows lists to be given meaningful names, which can improve the readability of the configuration. For example, it is conventional to define a domain list called `local_domains` for all the domains that are handled locally on a host, using a configuration line such as `domainlist local_domains = localhost:my.dom.example`

Named lists are referenced by giving their name preceded by a plus sign, so, for example, a router that is intended to handle local domains would be configured with the line `domains = +local_domains`

The first router in a configuration is often one that handles all domains except the local ones, using a configuration with a negated item like this: `dnslookup:`

```
driver = dnslookup
domains = ! +local_domains
transport = remote_smtp
no_more
```

The four kinds of named list are created by configuration lines starting with the words `domainlist`, `hostlist`, `addresslist`, or `localpartlist`, respectively. Then there follows the name that you are defining, followed by an equals sign and the list itself. For example: `hostlist relay_hosts = 192.168.23.0/24 : my.friend.example`  
`addresslist bad_senders = cdb;/etc/badsenders`

A named list may refer to other named lists: `domainlist dom1 = first.example : second.example`  
`domainlist dom2 = +dom1 : third.example`  
`domainlist dom3 = fourth.example : +dom2 : fifth.example`

Warning: If the last item in a referenced list is a negative one, the effect may not be what you intended, because the negation does not propagate out to the higher level. For example, consider: `domainlist dom1 = !a.b`  
`domainlist dom2 = +dom1 : *.b`

The second list specifies "either in the `dom1` list or `*.b`". The first list specifies just "not `a.b`", so the domain `x.y` matches it. That means it matches the second list as well. The effect is not the same as `domainlist dom2 = !a.b : *.b`

where x.y does not match. It's best to avoid negation altogether in referenced lists if you can.

Named lists may have a performance advantage. When Exim is routing an address or checking an incoming message, it caches the result of tests on named lists. So, if you have a setting such as `domains = +local_domains`

on several of your routers or in several ACL statements, the actual test is done only for the first one. However, the caching works only if there are no expansions within the list itself or any sublists that it references. In other words, caching happens only for lists that are known to be the same each time they are referenced.

By default, there may be up to 16 named lists of each type. This limit can be extended by changing a compile-time variable. The use of domain and host lists is recommended for concepts such as local domains, relay domains, and relay hosts. The default configuration is set up like this. **10.6 Named lists compared with macros**

At first sight, named lists might seem to be no different from macros in the configuration file. However, macros are just textual substitutions. If you write `ALIST = host1 : host2`  
`auth_advertise_hosts = !ALIST`

it probably won't do what you want, because that is exactly the same as `auth_advertise_hosts = !host1 : host2`

Notice that the second host name is not negated. However, if you use a host list, and write `hostlist alist = host1 : host2`  
`auth_advertise_hosts = !+alist`

the negation applies to the whole list, and so that is equivalent to `auth_advertise_hosts = !host1 : !host2`

#### 10.7 Named list caching

While processing a message, Exim caches the result of checking a named list if it is sure that the list is the same each time. In practice, this means that the cache operates only if the list contains no \$ characters, which guarantees that it will not change when it is expanded. Sometimes, however, you may have an expanded list that you know will be the same each time within a given message. For example: `domainlist special_domains = \`

```
  ${lookup{$sender_host_address}cdb{/some/file}}
```

This provides a list of domains that depends only on the sending host's IP address. If this domain list is referenced a number of times (for example, in several ACL lines, or in several routers) the result of the check is not cached by default, because Exim does not know that it is going to be the same list each time.

By appending `_cache` to `domainlist` you can tell Exim to go ahead and cache the result anyway. For example:  
`domainlist_cache special_domains = ${lookup{...`

If you do this, you should be absolutely sure that caching is going to do the right thing in all cases. When in doubt, leave it out. **10.8 Domain lists**

Domain lists contain patterns that are to be matched against a mail domain. The following types of item may appear in domain lists:

-

If a pattern consists of a single @ character, it matches the local host name, as set by the `primary_hostname` option (or defaulted). This makes it possible to use the same configuration file on several different hosts that differ only in their names.

-

If a pattern consists of the string `@[]` it matches any local IP interface address, enclosed in square brackets, as in an email address that contains a domain literal. In today's Internet, the use of domain literals is controversial.

-

If a pattern consists of the string `@mx_any` it matches any domain that has an MX record pointing to the local host or to any host that is listed in `hosts_treat_as_local`. The items `@mx_primary` and `@mx_secondary` are similar, except that the first matches only when a primary MX target is the local host, and the second only when no primary MX target is the local host, but a secondary MX target is. "Primary" means an MX record with the lowest preference value

&ndash; there may of course be more than one of them.

The MX lookup that takes place when matching a pattern of this type is performed with the resolver options for widening names turned off. Thus, for example, a single-component domain will not be expanded by adding the resolver's default domain. See the `qualify_single` and `search_parents` options of the `dnslookup` router for a discussion of domain widening.

Sometimes you may want to ignore certain IP addresses when using one of these patterns. You can specify this by following the pattern with `/ignore=<ip list>`, where `<ip list>` is a list of IP addresses. These addresses are ignored when processing the pattern (compare the `ignore_target_hosts` option on a router). For example: `domains = @mx_any/ignore=127.0.0.1`

This example matches any domain that has an MX record pointing to one of the local host's IP addresses other than 127.0.0.1.

The list of IP addresses is in fact processed by the same code that processes host lists, so it may contain CIDR-coded network specifications and it may also contain negative items.

Because the list of IP addresses is a sublist within a domain list, you have to be careful about delimiters if there is more than one address. Like any other list, the default delimiter can be changed. Thus, you might have: `domains = @mx_any/ignore=<;127.0.0.1;0.0.0.0 : \ an.other.domain : ...`

so that the sublist uses semicolons for delimiters. When IPv6 addresses are involved, it is easiest to change the delimiter for the main list as well: `domains = <? @mx_any/ignore=<;127.0.0.1;;;1 ? \ an.other.domain ? ...`

-

If a pattern starts with an asterisk, the remaining characters of the pattern are compared with the terminating characters of the domain. The use of `*&rdquo;` in domain lists differs from its use in partial matching lookups. In a domain list, the character following the asterisk need not be a dot, whereas partial matching works only in terms of dot-separated components. For example, a domain list item such as `*key.ex` matches `donkey.ex` as well as `cipher.key.ex`.

-

If a pattern starts with a circumflex character, it is treated as a regular expression, and matched against the domain using a regular expression matching function. The circumflex is treated as part of the regular expression. References to descriptions of the syntax of regular expressions are given in chapter 8.

Warning: Because domain lists are expanded before being processed, you must escape any backslash and dollar characters in the regular expression, or use the special `\N` sequence (see chapter 11) to specify that it is not to be expanded (unless you really do want to build a regular expression by expansion, of course).

-

If a pattern starts with the name of a single-key lookup type followed by a semicolon (for example, `&ldquo;dbm;&rdquo;` or `&ldquo;lsearch;&rdquo;`), the remainder of the pattern must be a file name in a suitable format for the lookup type. For example, for `&ldquo;cdb;&rdquo;` it must be an absolute path: `domains = cdb;/etc/mail/local_domains.cdb`

The appropriate type of lookup is done on the file using the domain name as the key. In most cases, the data that is looked up is not used; Exim is interested only in whether or not the key is present in the file. However, when a lookup is used for the `domains` option on a router or a `domains` condition in an ACL statement, the data is preserved in the `$domain_data` variable and can be referred to in other router options or other statements in the same ACL.

-

Any of the single-key lookup type names may be preceded by `partial<n>-`, where the `<n>` is optional, for example, `domains = partial-dbm;/partial/domains`

This causes partial matching logic to be invoked; a description of how this works is given in section 9.7.

-

Any of the single-key lookup types may be followed by an asterisk. This causes a default lookup for a key consisting of

a single asterisk to be done if the original lookup fails. This is not a useful feature when using a domain list to select particular domains (because any domain would match), but it might have value if the result of the lookup is being used via the `$domain_data` expansion variable.

If the pattern starts with the name of a query-style lookup type followed by a semicolon (for example, `&ldquo;nisplus;&rdquo;` or `&ldquo;ldap;&rdquo;`), the remainder of the pattern must be an appropriate query for the lookup type, as described in chapter 9. For example: `hold_domains = mysql;select domain from holdlist \ where domain = '$domain';`

In most cases, the data that is looked up is not used (so for an SQL query, for example, it doesn't matter what field you select). Exim is interested only in whether or not the query succeeds. However, when a lookup is used for the domains option on a router, the data is preserved in the `$domain_data` variable and can be referred to in other options.

If none of the above cases apply, a caseless textual comparison is made between the pattern and the domain.

Here is an example that uses several different kinds of pattern: `domainlist funny_domains = \`  
`@ : \`  
`lib.unseen.edu : \`  
`*.foundation.fict.example : \`  
`\N^[1-2]d{3}\.fict\.example$N : \`  
`partial-dbm;/opt/data/penguin/book : \`  
`nis;domains.byname : \`  
`nisplus;[name=$domain,status=local],domains.org_dir`

There are obvious processing trade-offs among the various matching modes. Using an asterisk is faster than a regular expression, and listing a few names explicitly probably is too. The use of a file or database lookup is expensive, but may be the only option if hundreds of names are required. Because the patterns are tested in order, it makes sense to put the most commonly matched patterns earlier. [10.9 Host lists](#)

Host lists are used to control what remote hosts are allowed to do. For example, some hosts may be allowed to use the local host as a relay, and some may be permitted to use the SMTP ETRN command. Hosts can be identified in two different ways, by name or by IP address. In a host list, some types of pattern are matched to a host name, and some are matched to an IP address. You need to be particularly careful with this when single-key lookups are involved, to ensure that the right value is being used as the key. [10.10 Special host list patterns](#)

If a host list item is the empty string, it matches only when no remote host is involved. This is the case when a message is being received from a local process using SMTP on the standard input, that is, when a TCP/IP connection is not used.

The special pattern `&ldquo;*&rdquo;` in a host list matches any host or no host. Neither the IP address nor the name is actually inspected. [10.11 Host list patterns that match by IP address](#)

If an IPv4 host calls an IPv6 host and the call is accepted on an IPv6 socket, the incoming address actually appears in the IPv6 host as `::ffff:<v4address>`. When such an address is tested against a host list, it is converted into a traditional IPv4 address first. (Not all operating systems accept IPv4 calls on IPv6 sockets, as there have been some security concerns.)

The following types of pattern in a host list check the remote host by inspecting its IP address:

If the pattern is a plain domain name (not a regular expression, not starting with `*`, not a lookup of any kind), Exim calls the operating system function to find the associated IP address(es). Exim uses the newer `getipnodebyname()` function when available, otherwise `gethostbyname()`. This typically causes a forward DNS lookup of the name. The result is compared with the IP address of the subject host.

If there is a temporary problem (such as a DNS timeout) with the host name lookup, a temporary error occurs. For example, if the list is being used in an ACL condition, the ACL gives a `&ldquo;defer&rdquo;` response, usually leading to a temporary SMTP error code. If no IP address can be found for the host name, what happens is described in section [10.14](#) below.

If the pattern is `&ldquo;@&rdquo;`, the primary host name is substituted and used as a domain name, as just described.

-

If the pattern is an IP address, it is matched against the IP address of the subject host. IPv4 addresses are given in the normal "dotted-quad" notation. IPv6 addresses can be given in colon-separated format, but the colons have to be doubled so as not to be taken as item separators when the default list separator is used. IPv6 addresses are recognized even when Exim is compiled without IPv6 support. This means that if they appear in a host list on an IPv4-only host, Exim will not treat them as host names. They are just addresses that can never match a client host.

-

If the pattern is "@", it matches the IP address of any IP interface on the local host. For example, if the local host is an IPv4 host with one interface address 10.45.23.56, these two ACL statements have the same effect:

```
accept hosts = 127.0.0.1 : 10.45.23.56
accept hosts = @
```

-

If the pattern is an IP address followed by a slash and a mask length (for example 10.11.42.0/24), it is matched against the IP address of the subject host under the given mask. This allows, an entire network of hosts to be included (or excluded) by a single item. The mask uses CIDR notation; it specifies the number of address bits that must match, starting from the most significant end of the address.

Note: The mask is not a count of addresses, nor is it the high number of a range of addresses. It is the number of bits in the network portion of the address. The above example specifies a 24-bit netmask, so it matches all 256 addresses in the 10.11.42.0 network. An item such as 192.168.23.236/31

matches just two addresses, 192.168.23.236 and 192.168.23.237. A mask value of 32 for an IPv4 address is the same as no mask at all; just a single address matches.

Here is another example which shows an IPv4 and an IPv6 network: recipient\_unqualified\_hosts = 192.168.0.0/16; \3ffe::ffff:836f:::/48

The doubling of list separator characters applies only when these items appear inline in a host list. It is not required when indirecting via a file. For example: recipient\_unqualified\_hosts = /opt/exim/unqualnets

could make use of a file containing 172.16.0.0/12  
3ffe::ffff:836f:::/48

to have exactly the same effect as the previous example. When listing IPv6 addresses inline, it is usually more convenient to use the facility for changing separator characters. This list contains the same two networks:

```
recipient_unqualified_hosts = <; 172.16.0.0/12; \
3ffe::ffff:836f:::/48
```

The separator is changed to semicolon by the leading "<;" at the start of the list. 10.12 Host list patterns for single-key lookups by host address

When a host is to be identified by a single-key lookup of its complete IP address, the pattern takes this form: net-<single-key-search-type>;<search-data>

For example: hosts\_lookup = net-cdb;/hosts-by-ip.db

The text form of the IP address of the subject host is used as the lookup key. IPv6 addresses are converted to an unabbreviated form, using lower case letters, with dots as separators because colon is the key terminator in lsearch files. [Colons can in fact be used in keys in lsearch files by quoting the keys, but this is a facility that was added later.] The data returned by the lookup is not used.

Single-key lookups can also be performed using masked IP addresses, using patterns of this form: net<number>-<single-key-search-type>;<search-data>

For example: net24-dbm;/networks.db

The IP address of the subject host is masked using <number> as the mask length. A textual string is constructed from the masked value, followed by the mask, and this is used as the lookup key. For example, if the host's IP address is 192.168.34.6, the key that is looked up for the above example is "192.168.34.0/24". IPv6 addresses are converted to a text value using lower case letters and dots as separators instead of the more usual colon, because colon is the key terminator in lsearch files. Full, unabbreviated IPv6 addresses are always used.

Warning: Specifying net32- (for an IPv4 address) or net128- (for an IPv6 address) is not the same as specifying just net- without a number. In the former case the key strings include the mask value, whereas in the latter case the IP address is used on its own.

### 10.13 Host list patterns that match by host name

There are several types of pattern that require Exim to know the name of the remote host. These are either wildcard patterns or lookups by name. (If a complete hostname is given without any wildcarding, it is used to find an IP address to match against, as described in the section 10.11 above.)

If the remote host name is not already known when Exim encounters one of these patterns, it has to be found from the IP address. Although many sites on the Internet are conscientious about maintaining reverse DNS data for their hosts, there are also many that do not do this. Consequently, a name cannot always be found, and this may lead to unwanted effects. Take care when configuring host lists with wildcarded name patterns. Consider what will happen if a name cannot be found.

Because of the problems of determining host names from IP addresses, matching against host names is not as common as matching against IP addresses.

By default, in order to find a host name, Exim first does a reverse DNS lookup; if no name is found in the DNS, the system function (gethostbyaddr() or getipnodebyaddr() if available) is tried. The order in which these lookups are done can be changed by setting the host\_lookup\_order option.

There are some options that control what happens if a host name cannot be found. These are described in section 10.14 below.

As a result of aliasing, hosts may have more than one name. When processing any of the following types of pattern, all the host's names are checked:

-

If a pattern starts with "/\*"; the remainder of the item must match the end of the host name. For example, \*.b.c matches all hosts whose names end in .b.c. This special simple form is provided because this is a very common requirement. Other kinds of wildcarding require the use of a regular expression.

-

If the item starts with "^"; it is taken to be a regular expression which is matched against the host name. For example, ^(a|b)\.c\.d\$

is a regular expression that matches either of the two hosts a.c.d or b.c.d. When a regular expression is used in a host list, you must take care that backslash and dollar characters are not misinterpreted as part of the string expansion. The simplest way to do this is to use \N to mark that part of the string as non-expandable. For example:  
sender\_unqualified\_hosts = \N^(a|b)\.c\.d\$\N : ....

Warning: If you want to match a complete host name, you must include the \$ terminating metacharacter in the regular expression, as in the above example. Without it, a match at the start of the host name is all that is required.

### 10.14 Behaviour when an IP address or name cannot be found

While processing a host list, Exim may need to look up an IP address from a name (see section 10.11), or it may need to look up a host name from an IP address (see section 10.13). In either case, the behaviour when it fails to find the information it is seeking is the same.

By default, Exim behaves as if the host does not match the list. This may not always be what you want to happen. To change Exim's behaviour, the special items +include\_unknown or +ignore\_unknown may appear in the list (at top level &ndash; they are not recognized in an indirected file).

-

If any item that follows `+include_unknown` requires information that cannot be found, Exim behaves as if the host does match the list. For example, `host_reject_connection = +include_unknown:*.enemy.ex`

rejects connections from any host whose name matches `*.enemy.ex`, and also any hosts whose name it cannot find.

If any item that follows `+ignore_unknown` requires information that cannot be found, Exim ignores that item and proceeds to the rest of the list. For example: `accept hosts = +ignore_unknown : friend.example : \`  
`192.168.4.5`

accepts from any host whose name is `friend.example` and from `192.168.4.5`, whether or not its host name can be found. Without `+ignore_unknown`, if no name can be found for `192.168.4.5`, it is rejected.

Both `+include_unknown` and `+ignore_unknown` may appear in the same list. The effect of each one lasts until the next, or until the end of the list.

Note: This section applies to permanent lookup failures. It does not apply to temporary DNS errors. They always cause a defer action (except when `dns_again_means_nonexist` converts them into permanent errors). 10.15 Host list patterns for single-key lookups by host name

If a pattern is of the form `<single-key-search-type>;<search-data>`

for example `dbm;/host/accept/list`

a single-key lookup is performed, using the host name as its key. If the lookup succeeds, the host matches the item. The actual data that is looked up is not used.

Reminder: With this kind of pattern, you must have host names as keys in the file, not IP addresses. If you want to do lookups based on IP addresses, you must precede the search type with `&quot;net-&rdquo;` (see section 10.12). There is, however, no reason why you could not use two items in the same list, one doing an address lookup and one doing a name lookup, both using the same file. 10.16 Host list patterns for query-style lookups

If a pattern is of the form `<query-style-search-type>;<query>`

the query is obeyed, and if it succeeds, the host matches the item. The actual data that is looked up is not used. The variables `$sender_host_address` and `$sender_host_name` can be used in the query. For example: `hosts_lookup = pgsql;\`  
`select ip from hostlist where ip='$sender_host_address'`

The value of `$sender_host_address` for an IPv6 address contains colons. You can use the `sg` expansion item to change this if you need to. If you want to use masked IP addresses in database queries, you can use the `mask` expansion operator.

If the query contains a reference to `$sender_host_name`, Exim automatically looks up the host name if has not already done so. (See section 10.13 for comments on finding host names.)

Historical note: prior to release 4.30, Exim would always attempt to find a host name before running the query, unless the search type was preceded by `net-`. This is no longer the case. For backwards compatibility, `net-` is still recognized for query-style lookups, but its presence or absence has no effect. (Of course, for single-key lookups, `net-` is important. See section 10.12.) 10.17 Mixing wildcarded host names and addresses in host lists

If you have name lookups or wildcarded host names and IP addresses in the same host list, you should normally put the IP addresses first. For example, in an ACL you could have: `accept hosts = 10.9.8.7 : *.friend.example`

The reason for this lies in the left-to-right way that Exim processes lists. It can test IP addresses without doing any DNS lookups, but when it reaches an item that requires a host name, it fails if it cannot find a host name to compare with the pattern. If the above list is given in the opposite order, the accept statement fails for a host whose name cannot be found,

even if its IP address is 10.9.8.7.

If you really do want to do the name check first, and still recognize the IP address, you can rewrite the ACL like this:

```
accept hosts = *.friend.example
accept hosts = 10.9.8.7
```

If the first accept fails, Exim goes on to try the second one. See chapter 39 for details of ACLs. 10.18 Address lists

Address lists contain patterns that are matched against mail addresses. There is one special case to be considered: the sender address of a bounce message is always empty. You can test for this by providing an empty item in an address list. For example, you can set up a router to process bounce messages by using this option setting: `senders = :`

The presence of the colon creates an empty item. If you do not provide any data, the list is empty and matches nothing. The empty sender can also be detected by a regular expression that matches an empty string, and by a query-style lookup that succeeds when `$sender_address` is empty.

Non-empty items in an address list can be straightforward email addresses. For example: `senders = jbc@askone.example : hs@anacreon.example`

A certain amount of wildcarding is permitted. If a pattern contains an `@` character, but is not a regular expression and does not begin with a semicolon-terminated lookup type (described below), the local part of the subject address is compared with the local part of the pattern, which may start with an asterisk. If the local parts match, the domain is checked in exactly the same way as for a pattern in a domain list. For example, the domain can be wildcarded, refer to a named list, or be a lookup: `deny senders = *@*.spamming.site:\`

```
*@+hostile_domains:\
bozo@partial-lsearch;/list/of/dodgy/sites:\
*@dbm;/bad/domains.db
```

If a local part that begins with an exclamation mark is required, it has to be specified using a regular expression, because otherwise the exclamation mark is treated as a sign of negation, as is standard in lists.

If a non-empty pattern that is not a regular expression or a lookup does not contain an `@` character, it is matched against the domain part of the subject address. The only two formats that are recognized this way are a literal domain, or a domain pattern that starts with `*`. In both these cases, the effect is the same as if `*@` preceded the pattern. For example: `deny senders = enemy.domain : *.enemy.domain`

The following kinds of more complicated address list pattern can match any address, including the empty address that is characteristic of bounce message senders:

```
-
```

If (after expansion) a pattern starts with `&ldquo;^&rdquo;`, a regular expression match is done against the complete address, with the pattern as the regular expression. You must take care that backslash and dollar characters are not misinterpreted as part of the string expansion. The simplest way to do this is to use `\N` to mark that part of the string as non-expandable. For example: `deny senders = \N^.*this.*@example.com$\N : \`

```
\N^d{8}.+@spamhaus.example$\N : ...
```

The `\N` sequences are removed by the expansion, so these items do indeed start with `&ldquo;^&rdquo;`; by the time they are being interpreted as address patterns.

```
-
```

Complete addresses can be looked up by using a pattern that starts with a lookup type terminated by a semicolon, followed by the data for the lookup. For example: `deny senders = cdb;/etc/blocked.senders : \`

```
mysql;select address from blocked where \
address='${quote_mysql:$sender_address}'
```

Both query-style and single-key lookup types can be used. For a single-key lookup type, Exim uses the complete address as the key. However, empty keys are not supported for single-key lookups, so a match against the empty address always fails. This restriction does not apply to query-style lookups.

Partial matching for single-key lookups (section 9.7) cannot be used, and is ignored if specified, with an entry being written to the panic log. However, you can configure lookup defaults, as described in section 9.6, but this is useful only for the &ldquo;\*@&rdquo; type of default. For example, with this lookup: `accept senders = lsearch*/;/some/file`

the file could contains lines like this: `user1@domain1.example`  
`*@domain2.example`

and for the sender address `nimrod@jaeger.example`, the sequence of keys that are tried is: `nimrod@jaeger.example`  
`*@jaeger.example`  
`*`

Warning 1: Do not include a line keyed by &ldquo;\*@&rdquo; in the file, because that would mean that every address matches, thus rendering the test useless.

Warning 2: Do not confuse these two kinds of item: `deny recipients = dbm*/;/some/file`  
`deny recipients = */dbm;/some/file`

The first does a whole address lookup, with defaulting, as just described, because it starts with a lookup type. The second matches the local part and domain independently, as described in a bullet point below.

The following kinds of address list pattern can match only non-empty addresses. If the subject address is empty, a match against any of these pattern types always fails.

If a pattern starts with &ldquo;@@&rdquo; followed by a single-key lookup item (for example, `@@lsearch*/;/some/file`), the address that is being checked is split into a local part and a domain. The domain is looked up in the file. If it is not found, there is no match. If it is found, the data that is looked up from the file is treated as a colon-separated list of local part patterns, each of which is matched against the subject local part in turn.

The lookup may be a partial one, and/or one involving a search for a default keyed by &ldquo;\*@&rdquo; (see section 9.6). The local part patterns that are looked up can be regular expressions or begin with &ldquo;\*@&rdquo;, or even be further lookups. They may also be independently negated. For example, with `deny senders = @@dbm;/etc/reject-by-domain`

the data from which the DBM file is built could contain lines like `baddomain.com: !postmaster : *`

to reject all senders except postmaster from that domain.

If a local part that actually begins with an exclamation mark is required, it has to be specified using a regular expression. In `lsearch` files, an entry may be split over several lines by indenting the second and subsequent lines, but the separating colon must still be included at line breaks. White space surrounding the colons is ignored. For example: `aol.com:`  
`spammer1 : spammer2 : ^[0-9]+$ :`  
`spammer3 : spammer4`

As in all colon-separated lists in Exim, a colon can be included in an item by doubling.

If the last item in the list starts with a right angle-bracket, the remainder of the item is taken as a new key to look up in order to obtain a continuation list of local parts. The new key can be any sequence of characters. Thus one might have entries like `aol.com: spammer1 : spammer 2 : >*`  
`xyz.com: spammer3 : >*`  
`*: \d{8}$`

in a file that was searched with `@@dbm*`, to specify a match for 8-digit local parts for all domains, in addition to the specific local parts listed for each domain. Of course, using this feature costs another lookup each time a chain is followed, but the effort needed to maintain the data is reduced.

It is possible to construct loops using this facility, and in order to catch them, the chains may be no more than fifty items long.

-

The `@@<lookup>` style of item can also be used with a query-style lookup, but in this case, the chaining facility is not available. The lookup can only return a single list of local parts.

Warning: There is an important difference between the address list items in these two examples: `senders = +my_list`  
`senders = *@+my_list`

In the first one, `my_list` is a named address list, whereas in the second example it is a named domain list. 10.19 Case of letters in address lists

Domains in email addresses are always handled caselessly, but for local parts case may be significant on some systems (see `careful_local_part` for how Exim deals with this when routing addresses). However, RFC 2505 (Anti-Spam Recommendations for SMTP MTAs) suggests that matching of addresses to blocking lists should be done in a case-independent manner. Since most address lists in Exim are used for this kind of control, Exim attempts to do this by default.

The domain portion of an address is always lowercased before matching it to an address list. The local part is lowercased by default, and any string comparisons that take place are done caselessly. This means that the data in the address list itself, in files included as plain file names, and in any file that is looked up using the `"@"` mechanism, can be in any case. However, the keys in files that are looked up by a search type other than `lsearch` (which works caselessly) must be in lower case, because these lookups are not case-independent.

To allow for the possibility of careful address list matching, if an item in an address list is the string `"+careful"`, the original case of the local part is restored for any comparisons that follow, and string comparisons are no longer case-independent. This does not affect the domain, which remains in lower case. However, although independent matches on the domain alone are still performed caselessly, regular expressions that match against an entire address become case-sensitive after `"+careful"` has been seen. 10.20 Local part lists

Case-sensitivity in local part lists is handled in the same way as for address lists, as just described. The `"+careful"` item can be used if required. In a setting of the `local_parts` option in a router with `careful_local_part` set false, the subject is lowercased and the matching is initially case-insensitive. In this case, `"+careful"` will restore case-sensitive matching in the local part list, but not elsewhere in the router. If `careful_local_part` is set true in a router, matching in the `local_parts` option is case-sensitive from the start. If a local part list is indirected to a file (see section 10.3), comments are handled in the same way as address lists — they are recognized only if the `#` is preceded by white space or the start of the line. Otherwise, local part lists are matched in the same way as domain lists, except that the special items that refer to the local host (`@`, `@[]`, `@mx_any`, `@mx_primary`, and `@mx_secondary`) are not recognized. Refer to section 10.8 for details of the other available item types.